Macro Programming through Bayesian Networks: Distributed Inference and Anomaly Detection

Marco Mamei
DISMI, Università di Modena e Reggio Emilia
Via Amendola 2, Reggio Emilia, Italy
marco.mamei@unimore.it

Radhika Nagpal
EECS, Harvard University
33 Oxford Street, Cambridge (MA), USA
rad@eecs.harvard.edu

Abstract

Macro programming a distributed system, such as a sensor network, is the ability to specify application tasks at a global level while relying on compiler-like software to translate the global tasks into the individual component activities. Bayesian networks can be regarded as a powerful tool for macro programming a distributed system in a variety of data analysis applications. In this paper we present our architecture to program a sensor network by means of Bayesian networks. We also present some applications developed on a microphone-sensor network, that demonstrate calibration, classification and anomaly detection.

1 Introduction

Future pervasive computing scenarios comprise a huge number of heterogeneous devices interacting with each other to achieve complex distributed applications. Sensor networks and networks of handheld computers could be employed in a variety of applications including environmental monitoring [7], navigation, and human interaction [8].

A key challenge is to provide powerful programming models to facilitate the development of applications in such dynamic and heterogeneous environments. The main conceptual difficulty is that we have direct control only on the individual components local activities, while the application task is often expressed at the global scale [11].

One promising research initiative in this direction is macro programming [2, 4, 5]. The idea is to be able to specify global application tasks to be achieved and leaving to a compiler or a distributed middleware the task of mapping these global tasks into individual components activities. To build these languages there are two fundamental challenges:

1. Devising a global language suitable for expressing tasks in a given class of applications.
2. Devising a set of distributed algorithms to map the language into the individual component activities.

In this paper we illustrate the role of Bayesian networks in this direction:

1. The graphical model used to create Bayesian networks can be used to represent a number of distributed applications from a high level perspective.
2. Inference mechanisms are easy to implement even on sensor boards and can map high-level Bayesian models into low level computations.

This approach is valuable in that it allows to apply macro programming to data analysis and inference tasks. In fact, a limitation of currently proposed macro programming approaches [2] is that they mainly focus on spatial issues only, e.g., how to let the components assume a specified *spatial* shape, how to select components in a *spatial* region. However, the concept of macro programming could be applied also to other kind of tasks. For example, in the next sections we will present how some applications based on distributed sensor fusion and not on spatial concepts only (e.g., distributed calibration, classification and anomaly detection) can be addressed in terms of Bayesian networks.

2 Bayesian Networks as a Global Programming Language

Bayesian networks are probabilistic graphical models. Specifically, a Bayesian network is a directed acyclic graph of nodes representing random variables, and edges representing dependence relations among the variables. Generally speaking we can regard an arc from a node A to a node B as indicating that A "causes" B (see Fig. 1). Bayesian networks provide a coherent framework for modeling a wide variety of knowledge [10].

The basic operation we can do with a Bayesian network is *inference*. Given the observation that some of the nodes

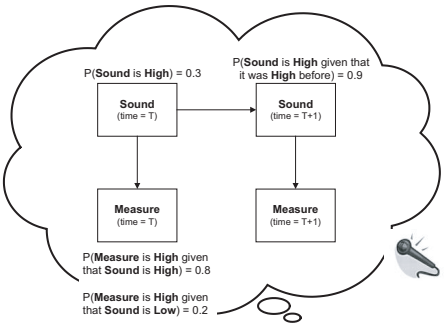

Figure 1. Simple Bayesian network, the actual sound probabilistically determines the measured sound.

are in a certain state, by relying on dependence relations, it is possible to compute the probability distributions of the other nodes. Several algorithms exist for performing inference both in centralized and distributed settings [1, 9, 10]. Inference allows one to perform three types of analyses:

1. **Prediction.** Compute the probability distribution of some nodes given the distribution of their *parents*. In other words, this is inference from "causes" to "effects". Considering the simple example in Fig. 1, prediction allows to answer the question: *Given that now the sound is high, how is it going to change in the future?* Looking at the figure, we can trivially predict that the sound will stay high with 0.9 probability.

2. **Diagnosis.** Compute the probability distribution of some nodes given the distribution of their *children*. This is inference from "effects" to "causes". Considering Fig. 1 example, this is answering: *Having measured that the sound is high, what are the chances it is actually high?* Using Bayes formula, it is 0.63.

3. **Anomaly detection.** A byproduct of the inference operation is the likelihood value, that is a measure of how well the observations being made fit the Bayesian network model. Anomalies and rare events can be detected on the basis of this value: the more the likelihood is low, the more the observation is anomalous - i.e., the observation does not fit the model.

These three operations form the core of many applications in several different pervasive computing scenarios. For example, the core task in almost all the context-aware computing applications is to understand high-level context information from low-level sensor readings [8]. A Bayesian network is a natural tool to perform such operations: it can be used to infer context situations from sensor measurements and predict context dynamics. For example, a Bayesian network relating high-level activities with sound measurements (e.g., driving the car "causes" car sound) can be used the other way to infer what the user is doing.

Another example is using a Bayesian Network to model the environment sensed by a sensor network. One could diagnose the overall environment condition and make prediction about future evolutions [3, 7]. For example, a microphone-sensor network could feed a Bayesian network with recorded sounds to track the location of the sound source – the location of a sound source "causes" a specific sound volume at sensors' location. This can be used the other way to infer where the source is located.

A key feature is that these applications can be specified easily at a global level. A global problem, e.g. understanding the pattern of sound in an area, can be represented naturally by a graphical model (see Fig. 2 top left). Our approach is to define global tasks by specifying a Bayesian network in which the probability distribution associated to some of the nodes can be based on values coming from remote sensors. Once this specification has been made, there are several algorithms that allow one to distribute the inference computation in an efficient way [1, 3, 7, 9]. In a sensor network scenario, each sensor computes the distribution of nodes in its Bayesian network and communicates that information to its neighbors (in the Bayesian network topology) to allow them to compute their distributions in turn[1]. This translation from global graphical representation to node computation can be automated (see Fig. 2 bottom).

2.1 Related Work

Recently, Bayesian networks have been fruitfully applied to several pervasive computing and sensor network applications. These applications provide good examples on the kind of tasks that can be achieved using Bayesian networks as modeling tools.

In [8] Bayesian networks are used to learn and infer transportation routines of a user provided with a GPS-enriched PDA. A Bayesian network running on a server, is constantly fed with the users' GPS traces. The server can then infer where the user is probably heading, which bus he has to ride and where he has to get off. The results are sent back to the PDA that can alert the user on missing stops, and unusual deviation from the usual track.

In [7] a distributed Bayesian network is used to calibrate a temperature sensor network. In this work, the Bayesian network encodes a model of the environment specifying how temperature should change in a given area. Compar-

[1] It is worth noting that in a lot of sensor network applications, the Bayesian network topology resembles the physical network. This is because the concepts in the Bayesian network tend to be affected and related to concepts "hosted" in neighbor sensors (e.g., the sound in one place in influenced only by sounds nearby).

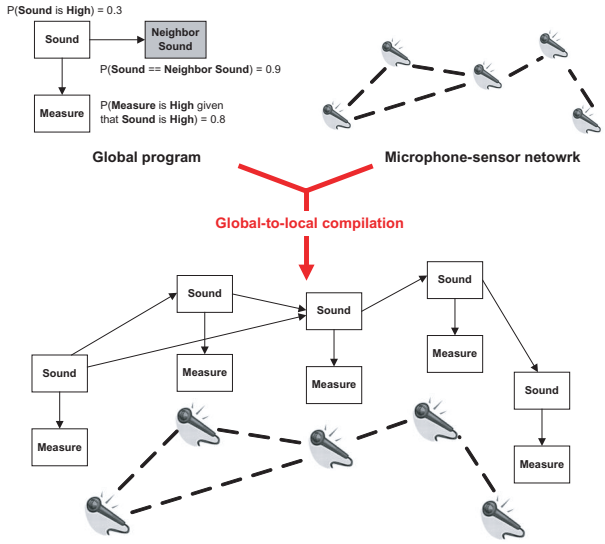

Figure 2. Bayesian network as a macro programming language

ing their actual temperature readings, the sensors are able to discover and correct biases in their readings.

Distributed Bayesian networks are used to localize distributed sensors [3]. The Bayesian network correlates sensors most likely location with the mutual distances between them. Exploiting this network each sensor is able to infer its most likely location.

This broad range of applications where Bayesian networks have been used motivates our work further.

3 Microphone Network

To test Bayesian networks as a macro programming language we performed some experiments on a microphone sensor network we instrumented in our lab. The current test-bed consists of wireless PDAs (HP IPAQ RX3700 – that come with an integrated microphone), and wireless laptops connected with low-cost microphones. In the experiments, we played different sounds from a speaker and let the nodes record and process the acquired sounds *online*.

We developed a framework in Java to create Bayesian networks and compute inference. Our framework allows one to distribute the Bayesian network across different nodes; using Java class loading capabilities, it is simple to let the nodes download the Bayesian network model so as to enable macro programming easily. Inference is based on particle filtering. The particle filter computation is performed in a distributed way using a mechanisms similar to the one proposed in [9]. Finally, we also implemented an EM-learning algorithm to learn in a unsupervised way the parameters of the Bayesian network[2]. The ideas underlying our implementation are based on [8]. Our implementation can be accessed from *http://sourceforge.net/projects/dbn-d*.

The current un-optimized, Java-based application running on the microphone-nodes is able to process signals online at 20-30 Hz (depending on the Bayesian network used to program the sensors). To meet this constraint, we downsampled the sound signal to the rate of 20 Hz. With such an infrastructure in place, we globally programmed the sensor network to achieve the applications described below.

3.1 Calibration

A typical problem in sensor network is *calibration*. Calibration is the process of forcing a sensor to conform to a given input/output mapping. In a lot of sensor network scenarios, calibration is challenging because of the huge number of sensors that are often not easily accessible. In these situations it would be valuable if sensors would be able to analyze the data they are collecting to discover and compensate for their own biases.

The general idea to achieve this autonomous calibration is to provide each sensor with a function of how the data it collects should relate to the data collected by other sensors around. For example, if two sensors should read the same values, but they systematically do not, then it is likely that they are biased. Sensors can then try to correct that bias by changing their internal parameters. If all the sensors perform this calibration procedure, they calibrate up to a constant factor that is then easy to be taken into account.

This calibration procedure can be easily realized by macro programming the sensor network with the Bayesian network depicted in Fig. 3. Each sensor runs a Bayesian network composed of three nodes. The measure (M) is modeled as a gaussian having a mean that is equal to the sum of the "real" sound (S) and of a possible bias (B). The bias (B) is initialized as a gaussian variable with 0 mean. The bias at time t, B(t) is modeled as a gaussian having a mean that is equal to B(t-1) and small covariance. This implies that the bias tends to remain constant in time. The sound (S) is influenced by the sounds collected by the other microphones around. In our experiments we assumed that all the microphones should probabilistically perceive the same sound, since they have been deployed close to each other in the same room. However any kind of relation between them could be applied depending on the context (i.e., location of microphones). In more detail, S(t) is modeled as a gaussian having a mean that is equal to the average of S(t-1) of all neighboring nodes. Taking the average of neigh-

[2]It is worth emphasizing that learning in Bayesian networks can be see a special case of diagnosis. We try to diagnose the most likely value of the parameters that condition the evidences [10]

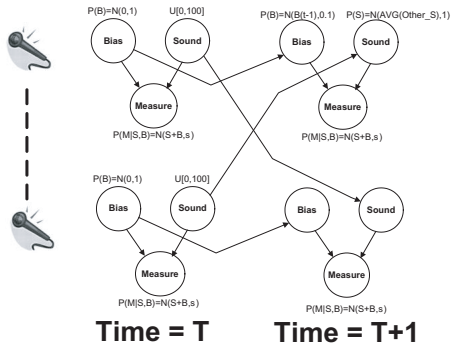

Figure 3. Bayesian network for sensor calibration

bor sounds is important to reduce the number of parameters needed to specify the sound variable.

In a first set of experiments (reported in Fig. 4) sensors perform calibration in the presence of a constant, but noisy sound. The figure illustrates the inference result of two sensors (other sensors display similar behavior).

- The square-plot (black) is the sound measured by the two sensors. The two sensors are biased, the top sensor perceives the sound with an average volume of 5, the bottom sensor perceives the same sound with an average volume of 15. The sound was played by an external speaker, microphones sample sound values at 20Hz and normalize the value in a 0 to 100 scale.
- The triangle-plot (blue) shows the most probable estimate of the bias parameter (please note that the graph has multiple scales, the right scale refers to the bias). It is easy to see that the bias of the top sensor tends to converge to a value near -5.5, while the bottom sensor converges to a value near 4.5. This means that the two sensors understand they are biased.
- The circle-plot (red) shows the most probable estimate of the sound parameter. It is possible to see that the calibration procedure succeeded and the two sensors believe that the actual sound is played at an average volume of 10.5.
- The diamond-plot (light-gray) shows the logarithm of the likelihood obtained during the inference process (rightmost scale). This parameter shows how likely the values perceived are with respect to the model. It is easy to see that at the beginning of calibration this value is low. This is because the model prescribes that the sound at different sensors should be perceived equal, and it is very unlikely that it is not. When the sensors calibrate the likelihood increases. This value is important because it allows the sensors to infer that the calibration has been completed successfully.

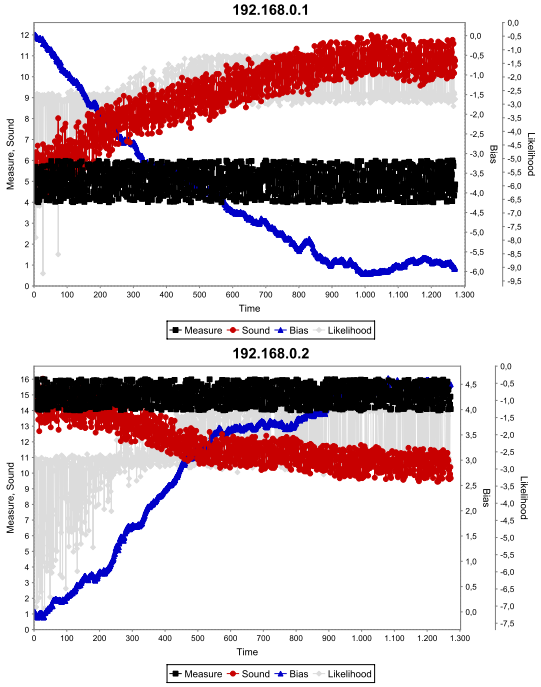

Figure 4. Result of calibration between two sensors for a simple flat signal

We conducted further experiments, not reported in the paper, that verified that the same result applies to a time-varying signal. The fact that calibration works also in presence of any kind of time-varying signal is important because we do not want to force a calibration signal in the environment, but we want the sensors to calibrate autonomously given the sounds they are naturally perceiving.

It is finally important to remark that in these experiments the network connectivity was rather reliable (TCP over WiFi). However this situation may not hold in other sensors' scenarios based on different network technologies. In any cases network glitches do not crash the system, in that nodes in the Bayesian network can update their distribution by simply disregarding remote information (e.g., a microphone may just think that the real sound is equal to the measured one). However, a large number of glitches (i.e., lost packets) can cause the calibration not to converge to a stable value, because nodes rarely compare their respective readings, and thus are not able to see their biases.

Anomaly Detection in Space. The experiment in Fig. 5 highlights an important property of Bayesian networks that allows it to detect anomalies in the sensor readings. Anoma-

lies are events that are unlikely given the model used to program the sensor network. In the experiment, we played a sound (between time 670 and 690) that was audible only by one sensor (see Fig. 5-top). The triangle-plot (black) represents a moving average of the likelihood with a sliding window of 10 samples. In all the experiments, the likelihood has been computed by simply multiplying together the wight of the individual particles as computed by the filtering algorithm [10]. Looking at the plot, it is easy to see that the sensor can realize that the sound is anomalous because of the low spike in the likelihood (the low spike is shifted forward in time from the anomalous sound because of the sliding window length). On the basis of the likelihood values the sensor computes an online statistical hypothesis test (t-test, H0=[log-likelihood=0], $p < 0.01$) to accept the readings as belonging to the model or to refuse them as anomalous (note that the null-hypothesis H0 is of a perfect match between the data and the model). Anomalous interval readings are readily identified and reported in the plot. In more detail, the system detects an anomaly interval beginning at the first anomalous reading and ending at the first next normal reading. With this in mind, it is interesting to see that this is anomaly detection in space. Sensors do not have a time-based model of the sound being played (in Fig. 3 there is not an arrow from S(t) to S(t+1)), but they have a space-based model, they know that sounds should be the same. Thus, the anomaly can be detected only in relation of what the spatially distributed sensors are hearing. With this regard it is interesting to see that also neighbor nodes can perceive an anomaly (see Fig. 5-bottom). This is because a sensor that measures a value different from the one being measured by one of its neighbors does not know whether the neighbor or the sensor itself is deviating from the usual pattern[3].

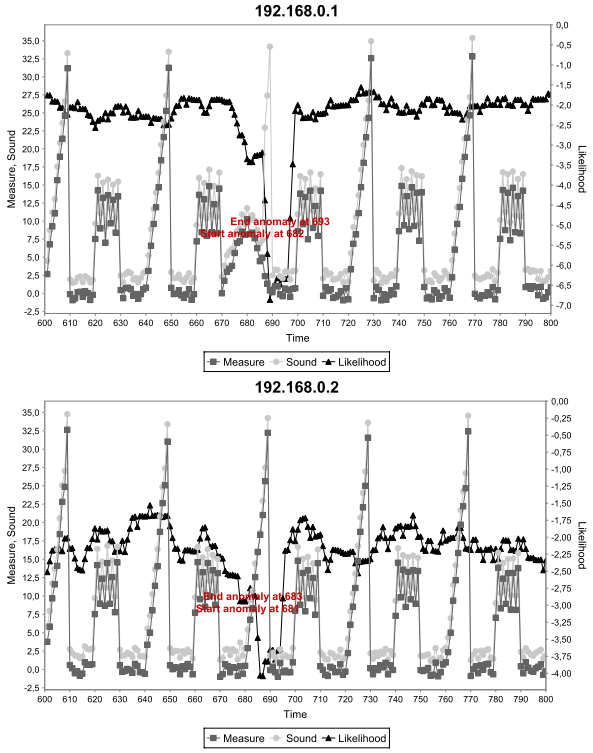

Figure 5. Anomaly detection in space

3.2 Classification

Another relevant application in sensor network is *classification*. Classifying the the data acquired by a sensor network is fundamental because it allows the network to communicate to a user a high-level description of what is happening in the sensed environment. This naturally facilitates the use of the acquired data and reduces enormously the required bandwidth.

To experiment with classification, we programmed each sensor with the Bayesian network depicted in Fig. 6. The class node (C) is a discrete random variable having a number of states equal to the number of classes used to classify the sounds. The amplitude node (A) is another discrete random variable that creates a discrete representation of the signal to be classified. Finally the sound node (S) is modeled as a gaussian node, and represents the measurement being taken.

All the nodes' conditional probability distributions are learned online using the unsupervised EM algorithm [8, 10]. This means that the first samples are used both for learning and classification, thus at first classification gives largely incorrect results. Once learning completes the learned parameters are substituted online to the previous parameters and classification becomes effective. In our experiments, the EM algorithm has been applied to a sliding-window evidence buffer composed of the last 30 samples, and it converges on average after 20 iterations. This amounts at about 30 seconds of on-line running conditions.

We want to emphasize that, although we used real hardware to run these experiments, the purpose of this classification is artificial. We are not trying to solve a specific application, we simply want to demonstrate that Bayesian networks, as a programming framework, can deal with this important application. This is also why we did not try to extract relevant features from the signal (this would have simplified classification a lot), but we try to classify the signal as it gets recorded. The result of classification is depicted in Fig. 7.

- The circle plot (dark gray) represents the sound being measured by the sensors. The sound was played by

[3] Actually, if the network is dense enough, these "false" anomalies do not arise since an anomalous neighbor value would be averaged out.

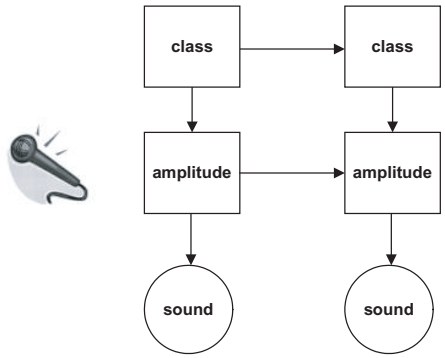

Figure 6. Bayesian network for sensor classification

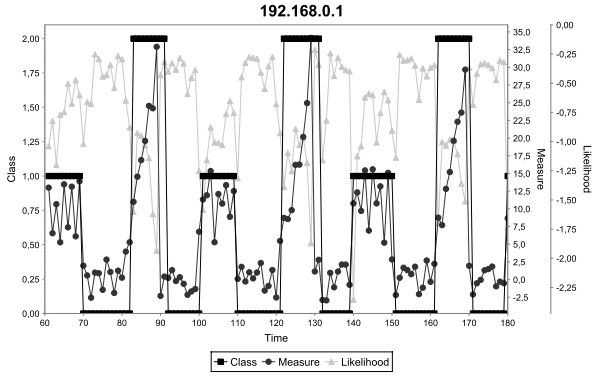

Figure 7. Result of the classification process

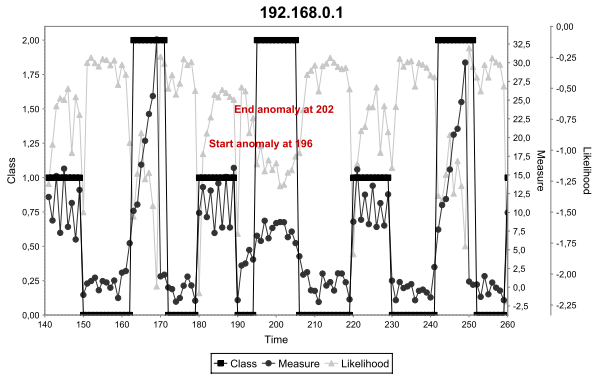

Figure 8. Anomaly detection in time

the speaker that repeatedly played three sounds: a 0-volume constant sound, a sound that increases from a volume of 0 to a volume of 30 and then goes back to 0, and an oscillating sound between 7 and 15 volume.

- The square plot (black) represents the result of the classification process. It is easy to see that the classification works very efficiently. It classifies the 0-volume sound with class 0, the oscillating sound with class 1, and the increasing sound with class 2. It is worth noting that, since the classification is unsupervised, it is not possible to decide *a priori* which signal will be classified by which class. However, once classification is correct, establishing this matching becomes trivial.

- The triangle plot (light gray) is the likelihood of the classification. It is actually a measure of how confident the sensor is about its classification judgment. It is interesting to notice that every time the class changes, the likelihood fall down sharply. The reason for this is twofold. On the one hand, when only few samples of the new sound are presented, it is difficult to assess to which class the signal will belong. On the other hand, since each sound signal lasts for a rather long time, the EM algorithm learns that the class node tends to remain constant to a given class, and it is unlikely that it changes.

Anomaly Detection in Time. In the experiment depicted in Fig. 8 we tried to verify if the Bayesian network is able to identify anomalies in time (i.e., unusual patterns). This is interesting in that it complements the detection of the anomalies in space previously described. In the figure, it is possible to see an anomalous pattern in the time interval 190-210. The triangle plot (light gray) shows the log likelihood of the classification. Similarly as before, sensors conduct a t-test (H0=[log-likelihood=0], $p < 0.01$) on the moving average of the likelihood (with a sliding window of 15 samples) to accept the readings as belonging to the model or to refuse them as anomalous. Anomalous interval readings are readily identified and reported in the plot.

3.3 Simultaneous Calibration and Classification

An important advantage of using Bayesian networks as a macro programming language is that their graphical model naturally supports high-level composition. For example the calibration and classification networks can be easily integrated by merging together the two sound nodes. In this way, the network calibrates and classifies at the same time. This naturally leads to some advantages:

1. In general, composition of Bayesian networks leads to a modular approach in macro programming. This is useful in that modules can be tested and learned separately and then tied together. Learning the modules separately can greatly ease the learning process and improve performance [6].

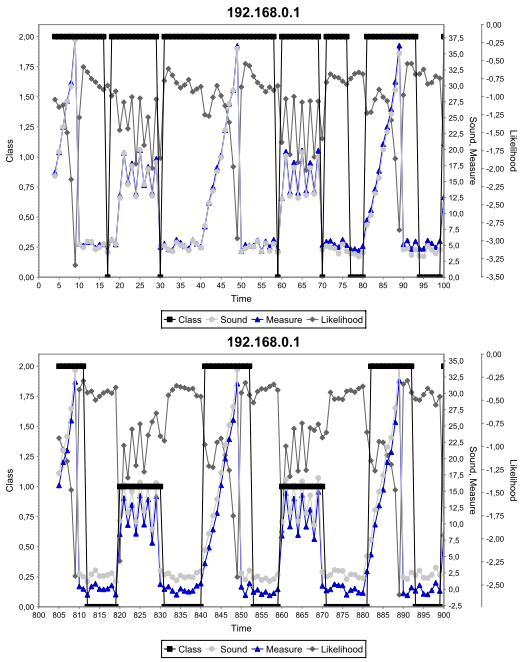

Figure 9. Simultaneous Calibration and classification

2. By combining together calibration and classification, it is possible to detect both spatial and temporal anomalies. Moreover, it could also be possible to detect space-time anomalies, i.e., signals that appear normal in time and space separately, but not if considering space and time relations together. We still did not experiment with such a situation, but if this idea will be confirmed, it would be an interesting "emergent" property of the model [11].

The result of the combined classification and calibration is depicted in Fig. 9. Both the two graphs refer to the same sensor. The top graph has been recoded at the beginning of the experiment. It is possible to see that un-calibrated biased measures trash the performance of the classifier. This is rather obvious, in that the classifier receives a sound with a volume different from the one used in the learning process. The bottom graph has been recorded after some time. Here calibration compensated the bias and classification provides again good results.

4 Conclusion

In this paper we illustrated the idea of using Bayesian networks as a macro programming language to program distributed systems like sensor networks. While most of the proposals in macro programming languages focus on spatial issues only (e.g., select components in a specified *spatial* region), Bayesian networks can deal with tasks not directly related to spatial concepts, but related to data analysis and inference (e.g., let sensors calibrate and classify signals).

Acknowledgments. Work supported by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission.

Nagpal is also supported by a Microsoft Research Faculty Fellowship.

References

[1] C. Crick and A. Pfeffer. Loopy belief propagation as a basis for communication in sensor networks. In *Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, San Francisco (CA), USA, 2003.

[2] S. Hadim and N. Mohamed. Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 7(3), 2006.

[3] A. Ihler, J. Fisher, R. Moses, and A. Willsky. Nonparametric belief propagation for self-calibration in sensor networks. *IEEE Journal of Selected Areas in Communication*, 23(4):809 – 819, 2005.

[4] R. Nagpal. Programmable self-assembly using biologically-inspired multiagent control. In *Joint Conference on Autonomous Agents and Multi-Agent Systems*. ACM Press, Bologna, Italy, 2002.

[5] R. Nagpal and M. Mamei. Engineering amorphous computing systems. In *Methodologies and Software Engineering for Agent Systems*. Kluwer Academic Publishing, 2004.

[6] N. Oliver and E. Horvitz. A comparison of hmms and dynamic bayesian networks for recognizing office activities. In *International Conference on User Modeling*. Edinburgh, UK, 2005.

[7] M. Paskin, C. Guestrin, and J. McFadden. A robust architecture for inference in sensor networks. In *International Symposium on Information Processing in Sensor Networks*. ACM Press, Los Angeles (CA), USA, 2005.

[8] D. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring high-level behavior from low-level sensors. In *International Conference on Ubiquitous Computing*. ACM Press, Seattle (WA), USA, 2003.

[9] M. Rosencrantz, G. Gordon, and S. Thrun. Decentralized sensor fusion with distributed particle filters. In *Conference on Uncertainty in AI*. Acapulco, Mexico, 2003.

[10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[11] F. Zambonelli and V. Parunak. Towards a paradigm change in computer science and software engineering: a synthesis. *The Knowledge Engineering Review*, 18(4), 2004.